ONTOLOGY FOR PIXEL PROCESSING

Zhengbo Zhou
668546

Final Graduate Thesis submitted to the Department of Computer Science, University of Bridgeport. Supervised by Prof. Rao. Submitted on June 29, 2005.

Ontology for Pixel Processing

1. Abstract

For all kinds of output devices, such as monitors, printers etc, the most important thing is to show the right information to the user. Pixel is the basic element both on screen and materials printed with. And, as a result pixel processing is the basic technique to make the output correct, precise, and suitable to use on different occasions. Pixel processing solves operations on each pixel of the image, which is for the pixel matrices of that image, so that the image would have different appearance.

Ontology is about the exact description of things and their relationships. It is an old study of philosophy from ancient Greece. As the study of artificial intelligence keeps growing, the concept of ontology has been in use more and more in the formalization of knowledge in terms of classes, properties, instances and relations [1].

This paper mainly discusses how to build ontology of pixel processing with OWL. Actually, it is focused on how to describe pixel processing and its functions or operations in an understandable way by computer. With such description, it is possible to improve the development of pixel processing and the sharing of its knowledge both between people and machines. That is from the Natural Language Processing point of view. And also, in the future, it provides a base for intelligent agent to implement pixel processing by understanding such kind of definition and description directly through its knowledge base built up with such ontology. In other words, that may realize the automatic program or program analysis.

Key words:
Pixel, Pixel Processing, Ontology, OWL, Description

2. Background

2.1 Pixel Processing

Pixel: (**PIX** [picture] **EL**ement) The smallest addressable unit on a display screen. The higher the pixel resolution (the more rows and columns of pixels), the more information can be displayed [3].

The definition is highly context sensitive. For example, we can speak of pixels in a visible image (e.g. a printed page) or pixels carried by one or more electronic signal(s), or represented by one or more digital value(s), or pixels on a display device. This list is not exhaustive and depending on context there are several synonyms which are accurate in particular contexts, e.g. pel, sample, bytes, bits, dots, spots, superset, triad, stripe set, window, etc [3]. We can also speak of pixels in the abstract, in particular when using pixels as a measure of resolution, e.g. 2400 pixels per inch or 640 pixels per line. Dots

are often used to mean pixels, especially by computer sales and marketing people, and gives rise to the abbreviation DPI or dots per inch.

Note that a pixel may be comprised of sub-parts or sub pixels. For example a pixel on a color display may be composed of red, green and blue sub-parts (sub-pixels, sub-pels, etc.) the three of which may be referred to as a triad. A pixel in a video signal may be composed of RGB parts or Y, R-Y, B-Y or Y, I, Q, or Y, C, M or subcarrier modulated Y or composite video or separate signals such as separate ones of the various three sub-pixels above. Many unskilled people, and sometimes skilled people, incorrectly use pixel and image element interchangeably, or use pixel to refer to sub-parts. Unskilled people don't know any better and the skilled people know better but because the meaning is clear from the context do so anyway. Many dictionaries also get it wrong [3].

Typical pixels we are concerned with in laser printers are those made up of sub-pels in the screening processes, those made up of yellow, cyan and magenta sub-pels in color printing and those which are simply dots of black toner in black and white printers. Typical pixels we are concerned with in television systems are the samples of composite video signals (a single digital value having Y and color subcarrier components) those carried by three electronic signals or three digital values, either Y, R-Y, B-Y or R, G, B depending on where in the TV we are looking and those displayed on the TV screen which are made up of R, G and B color sub-pixels. Note that Y, R-Y and B-Y values are often carried as two electronic signals in television applications, Y in one and time multiplexed R-Y, B-Y in the other [3].

Image element is a broader term than pixels and is also highly context sensitive. Image elements includes both complete pixels as well as those various sub-parts of pixels and other elements of images which are not pixel related such as DCT coefficients. For example, it is correct to say that the red part of an RGB pixel is an image element but it is not normally considered correct to refer to the red part as a pixel itself (although persons who are not skilled in the television industry often do).

In storage, pixels are made up of one or more bits. The greater this "bit depth," the more shades or colors can be represented. The most economical system is monochrome, which uses one bit per pixel (on/off). Gray scale and color displays typically use from 4 to 24 bits per pixel, providing from 16 to 16 million colors.

On screen, pixels are made up of one or more dots of color. Monochrome and gray scale systems use one dot per pixel. For monochrome, the dark pixel is energized light. For gray scale, the pixel is energized with different intensities, creating a range from dark to light. Color systems use a red, green and blue dot per pixel, each of which is energized to different intensities, creating a range of colors perceived as the mixture of these dots. Black is all three dots dark, white is all dots light.

2.2 Pixel processing functions

Pixel-based image processing includes the following
General pixel based manipulation including transendental functions are carried out by the following routines. For multiple image manipulation the image type returned is determined by the function im_sup_vtype

Vartype im_sup_vtype(Vartype vtype1, Vartype vtype2)
which ensures no truncation of the result due to casting (truncation may still occur for numbers beyond the normal range of number representation).
Single image operation [2]:

Imrect *im_add(double k,Imrect *im)
Add the constant k to all pixels in the specified image. For the case of complex images only the real component is modified.

void im_pixf_dec(Imrect * image, int i, int j)
Decrement pixel (i, j) in image. Pixels outside image are ignored.

Imrect *im_times(double k, Imrect * im)
Returns an image with pixels scaled by of their input values.

Imrect *im_minus(Imrect * im)
Inverts the sign of all pixel values returning a new image.

Imrect *im_sqr(Imrect * im)
Compute the sqaure of each image pixel value, all values returned as float_v except complex_v, which is returned as complex_v.

Imrect *im_log(Imrect * im)
Returns an image of natural logariths for all pixels while maintaining sign. The function is a direct inverse of im_exp(). All images are returned as type float_v except for type complex_v which is returned as complex_v.

Imrect *im_sqrt(Imrect * im)
Returns an image with each pixel the square-root of the initial grey level value. The sign of the original pixel is preserved, for true complex treatment of negative values use imz_sqrt().

Imrect *im_quad(im)
Returns an image which has been vertically and horizontally doubled to produce boundary continuity. Intended for use before Fourier transform operations to remove Gibbs oscillations after deconvolution.

Imrect *im_sin(Imrect * im)
Returns an image with each pixel given by the trigonometric sine of the initial grey level data.

Double image operation[2]:

Imrect *im_diff(Imrect * im1, Imrect * im2)
Subtract the pixels contained in two images and return the difference image.

Imrect *im_sum(Imrect * im1, Imrect * im2)
Returns an image whose pixels are the sum of those in the input images within the common region of interest.

Imrect *im_prod(Imrect * im1, Imrect * im2)
Returns an image with values given by the product of the pixels in the images im1 and im2.

Imrect *im_div(Imrect * im1, Imrect * im2, double thresh, double val)
Divide the pixels contained in image im1 by those contained in image im2. Numerical stability is maintained by preventing the denominator reducing below a value of when it's absolute value is less than .

Imrect *im_combine(Imrect * im1, Imrect * im2, void *(*func) (), void *data)
Returns an image with pixels computed from the specified input image pixels operated upon by the specified function.

Imrect *im_fpp_combine(Imrect * im1, Imrect * im2, void *(*func) (), void *data)
General purpose routine for operating on two images of types float_v and ptr_v with the specified function.

Still, there are a lot of operations or processing not listed here, since pixel processing is still developing new functions to satisfy different requirements. For example, pixel processing also deals with 3-dimensional computer graphic operations. And moreover, different companies will develop different algorithms and function to fit their own display products.

2.3 Ontology and OWL [1]

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

2.3.1 Ontology

As already mentioned, ontology is about the exact description of things and their relationships. For knowledge, ontology is about the exact description of the representation of the knowledge itself, as well as the relationships among different

categories of the knowledge. And moreover, for the web, ontology is about the exact description of web information and relationships between web information.

2.3.2 OWL: Development and Features

OWL is a Web Ontology Language, which is built on top of RDF – Resource Definition Framework and written in XML. It is a part of Semantic Web Vision, and was designed to be interpreted by computers, but not for being read by people. OWL became a W3C (World Wide Web Consortium) Recommendation in February 2004. The OWL Web Ontology Language is a language for defining and instantiating *Web ontologies*, and, OWL ontology may include the descriptions of classes, properties, and their instances. Given such ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but *entailed* by the semantics [1].

One of the effective approaches to solve the data exchange among different computers via the computer networks is using XML – eXtensible Markup Language. Since HTML is to solve how the data appears in front of people, it does not make different computers different systems to "know" the data, however, the documents written in XML can describe what data is and be shared and exchanged among different systems. XML provides a syntax for structured documents, however it imposes no semantic constraints on the meaning of the document. Following this, the W3 Consortium implemented the ideas of an ontology when creating RDF – the Resource Definition Framework. RDF is a data model for objects and relations between them, providing a simple semantic for this data model. RDF uses XML syntax to describe objects and relations in the data model.
After that, RDFS was developed to describe properties and classes of RDF resources, with a semantic for creating hierarchies of such objects and classes and thus providing the means for generalization.

RDFS is considered to be an ontology language, containing classes and properties and being aware of concepts of range and domain, as well as having the ability describe subclasses and superclasses. However, for implementing the Semantic Web RDFS is not quite optimal as it lacks the features necessary to describe resources in sufficient detail. As Santtu Toivonen concludes in his paper "Using RDF(S) to Provide Multiple Views into a Single Ontology" , RDFS is suitable for providing the means for an ontology that characterizes some environment, no matter how abstract. RDFS alone, however, suffers from its dependence on domain-specific and case-specific details. RDFS suffers from an expressive inadequacy and it lacks a number of important relations between classes such as equivalence and disjointedness, as well as cardinality and characteristics of properties.

To solve the problem RDFS brings as mentioned above, two languages were developed almost concurrently. OIL (Ontology Inference Layer) in Europe and DAML (DARPA Agent Markup Language) in U.S.. Both of them are based on top of RDFS. After submitted the combination of the two—DAML+OIL to W3 Consortium for standardization, OWL –the Ontology Web Language was born as a new W3C standard language. OWL is layered on top of RDFS, using its syntax for expressing ontological

primitives such as Class, Relation, Subclass etc. In addition OWL adds a much richer set of its own primitives, such as transitivity, cardinality, disjunction etc., as well as characteristics of properties like symmetry, richer typing of properties (e.g. nonNegativeInteger), and enumerated classes. As a result, OWL has more facilities for expressing meaning and semantics than XML, RDF (S), and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web.
The main purpose of OWL can be concluded as below:

1, Formalize a domain by defining classes and properties of those classes,
2, Define individuals and assert properties about them, and
3, Reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language.

2.3.3 The Species of OWL

All this as lead to a set of requirements that may seem incompatible: efficient reasoning support and convenience of expression for a language as powerful as a combination of RDF Schema with a full logic.

Indeed, these requirements have prompted W3C's Web Ontology Working Group to define OWL as three different sublanguages, each of which is geared towards fulfilling different aspects of these incompatible full set of requirements:

OWL Full: The entire language is called OWL Full, and uses all the OWL languages primitives. It also allows to combine these primitives in arbitrary ways with RDF and RDF Schema. The advantage of OWL Full is that it is fully upward compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL Full document, and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion. The disadvantage of OWL Full is the language has become so powerful as to be undecidable, dashing any hope of complete (let alone efficient) reasoning support. [3]

OWL DL: In order to regain computational efficiency, OWL DL (Description Logic) is a sublanguage of OWL Full which restricts the way in which the constructors from OWL and RDF can be used. The advantage of this is that it permits efficient reasoning support. The disadvantage is that we loose full compatibility with RDF: an RDF document will in general have to be extended in some ways and restricted in others before it is a legal OWL DL document. Conversely, every legal OWL DL document is still a legal RDF document.

OWL Lite: An ever further restriction limits OWL DL to a subset of the language constructors. For example, OWL Lite excludes enumerated classes, disjointness statements and arbitrary cardinality (among others). The advantage of this is a language that is both easier to grasp (for users) and easier to implement (for tool builders). The disadvantage is of course a restricted expressivity.

There are strict notions of upward compatibility between these three sublanguages:
Every legal OWL Lite ontology is a legal OWL DL ontology.

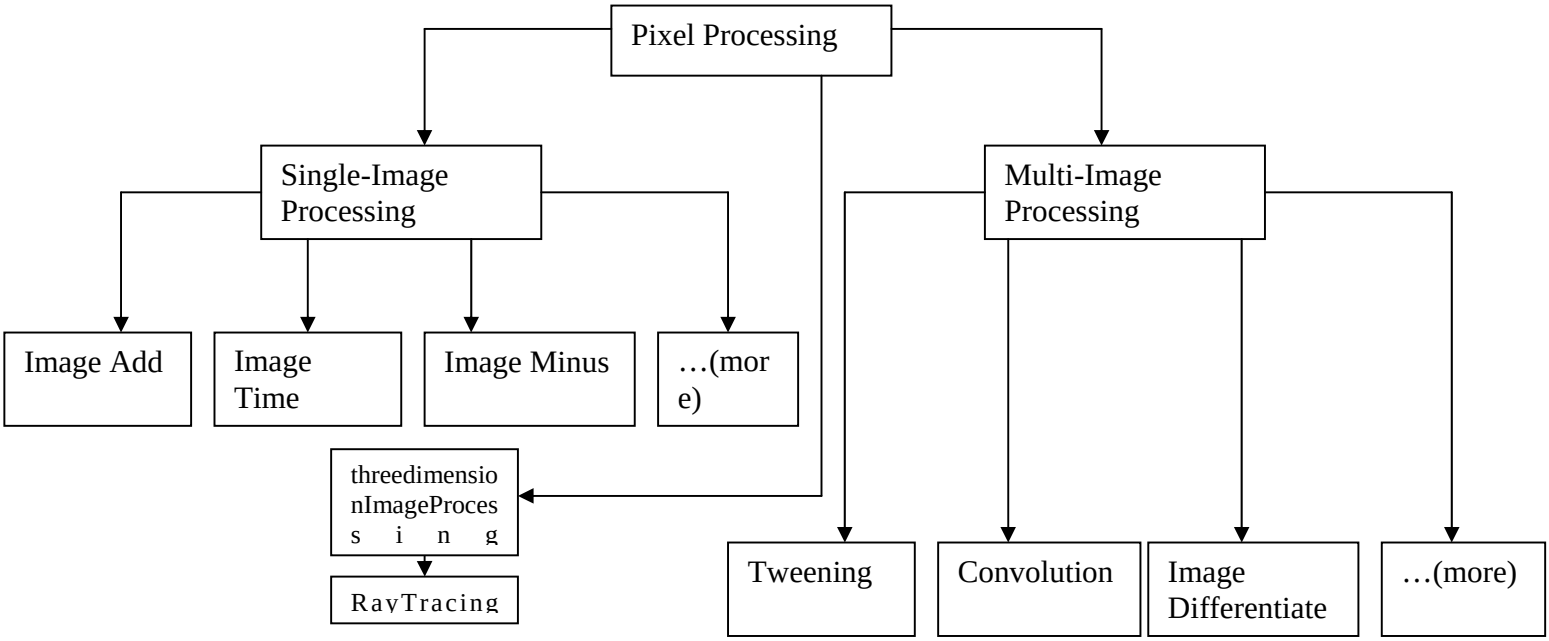
Every legal OWL DL ontology is a legal OWL Full ontology.
Every valid OWL Lite conclusion is a valid OWL DL conclusion.
Every valid OWL DL conclusion is a valid OWL Full conclusion.

3. Pixel Processing Ontology Design

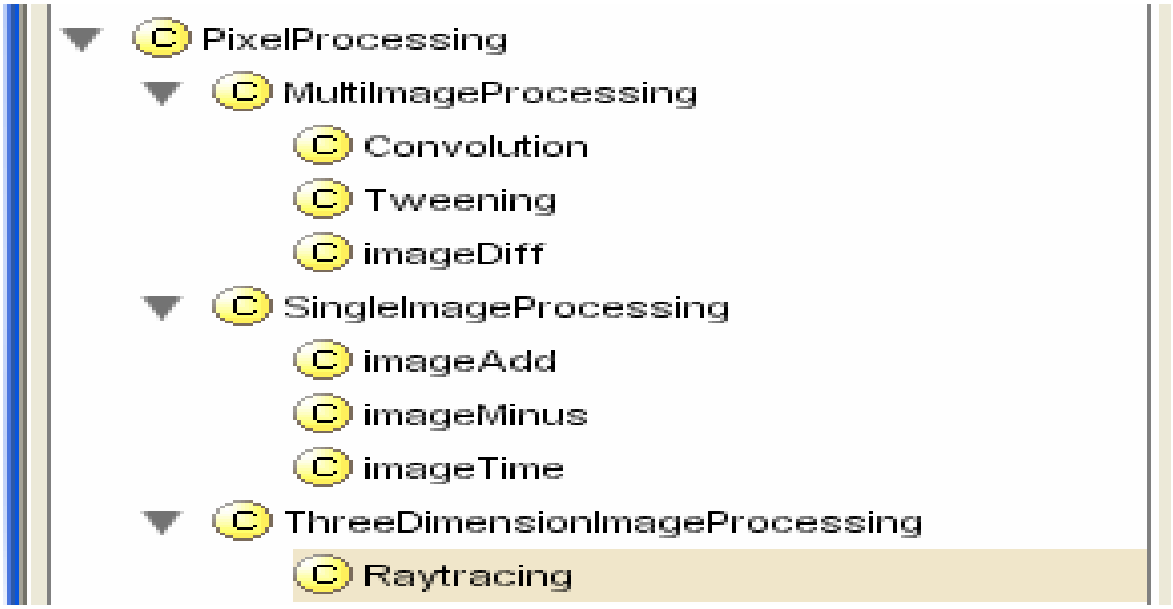
From the perspective of ontology, we can build up a class hierarchy of pixel processing and its related concepts.

3.1 Class analysis and hierarchy building:

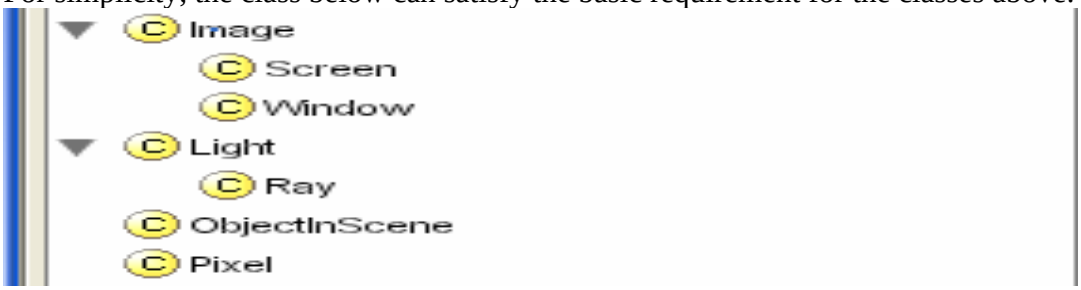
According to the basic functions (operations) provided in Background part and some special pixel processing operations, here is the basic class hierarchy model:



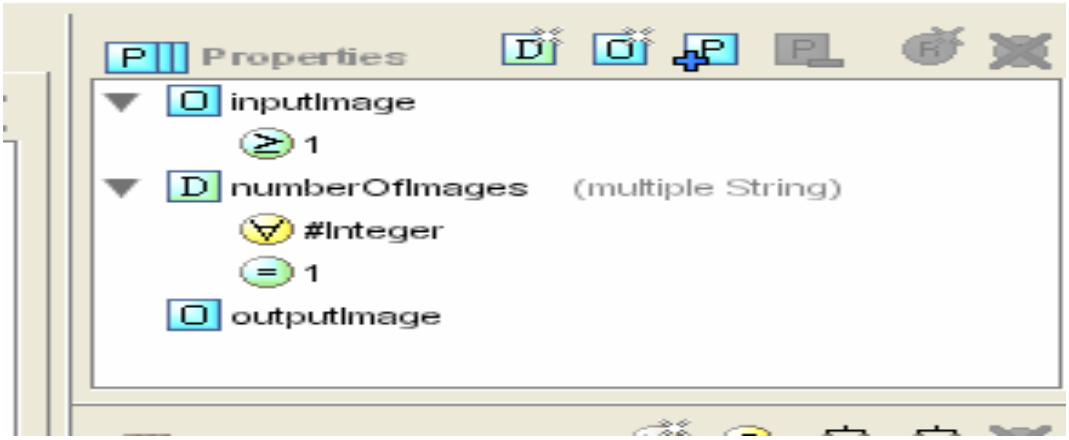
To implement this basic hierarchy in OWL easily, Protégé 3.0 Beta version has been used to generate the OWL script and also a graphic interface.
In protégé 3.0, we can build the class hierarchy as follows:



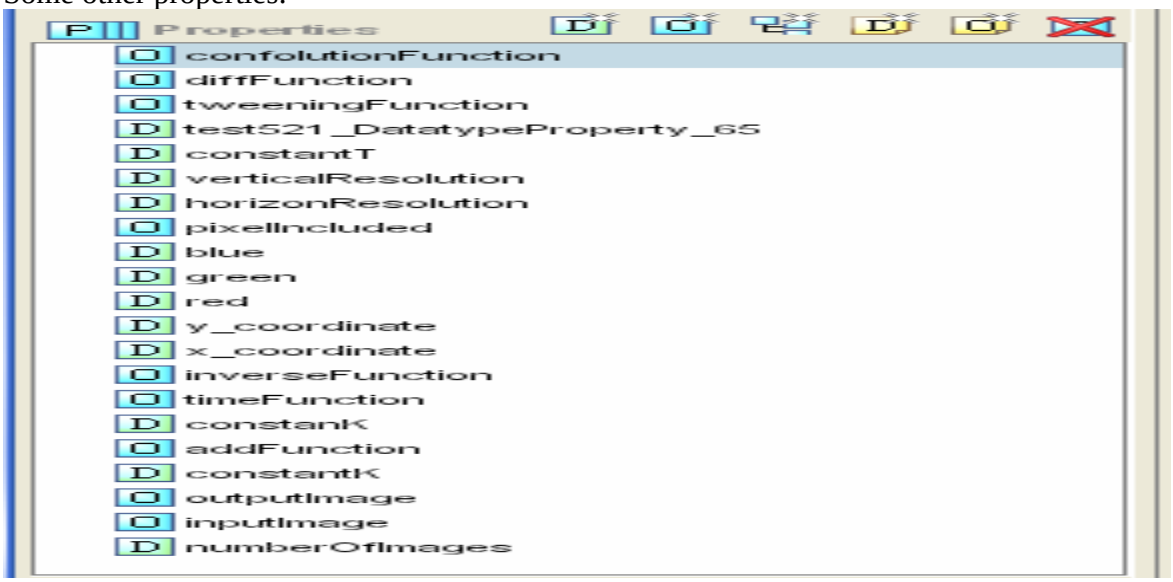
We can also have the ontology for related concepts
Pixel, Sub-Pixel, Megapixel, BitDepth, ColorSystem, etc, within which we can build a large knowledge base for the domain of computer graphics.
For simplicity, the class below can satisfy the basic requirement for the classes above.



3.2 Property (slots) analysis:
Properties should be included:
Pixel Processing: input (image), output (image), number of images

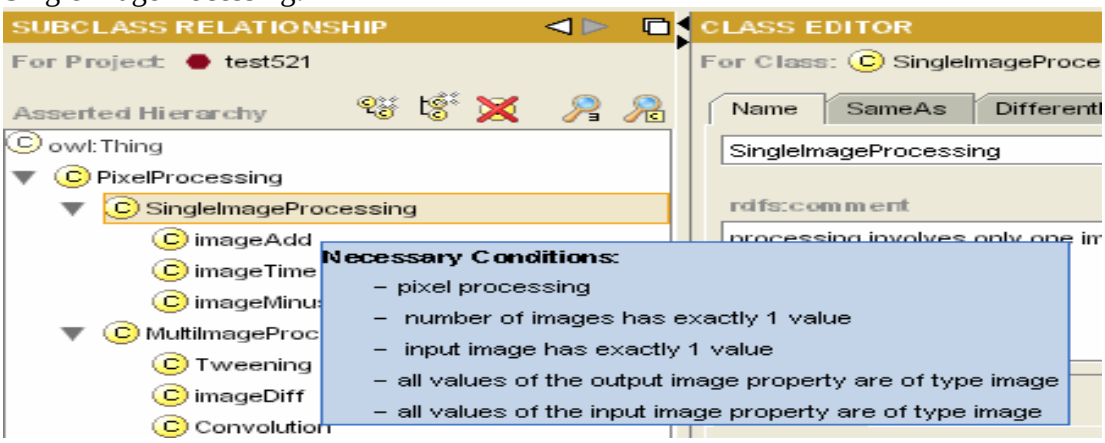


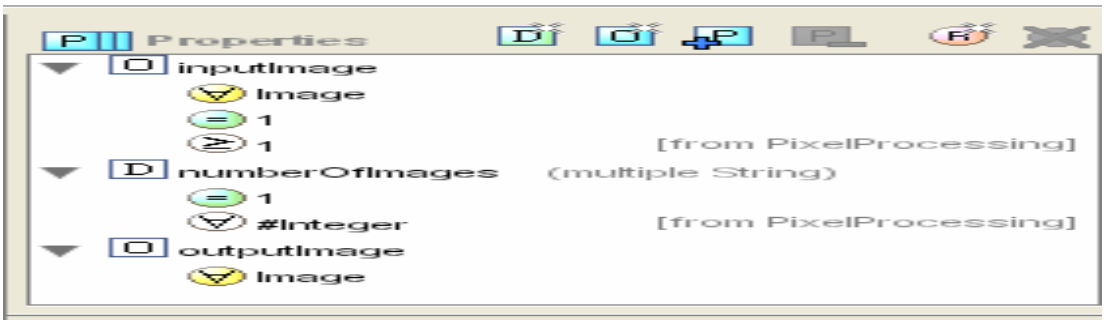
Some other properties:



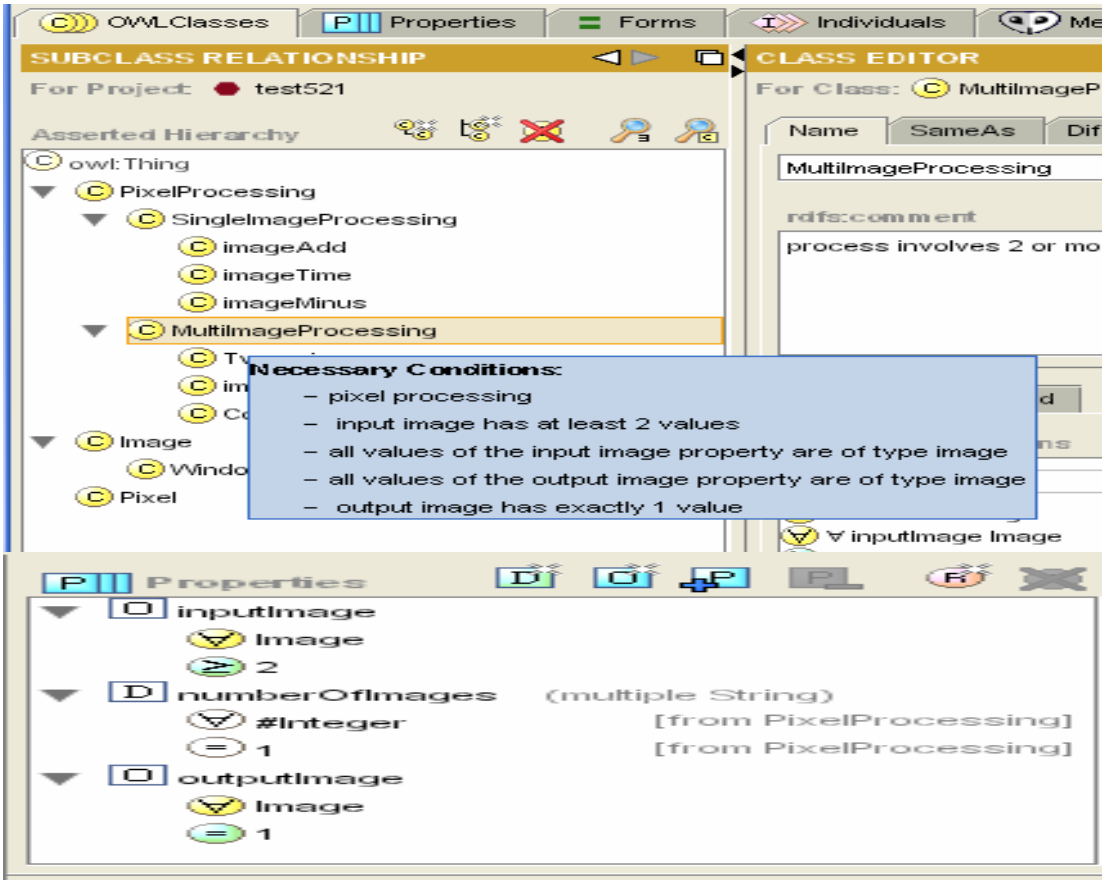
For each class we built in Protege 3.0, here is the property list:

SingleImageProcessing:

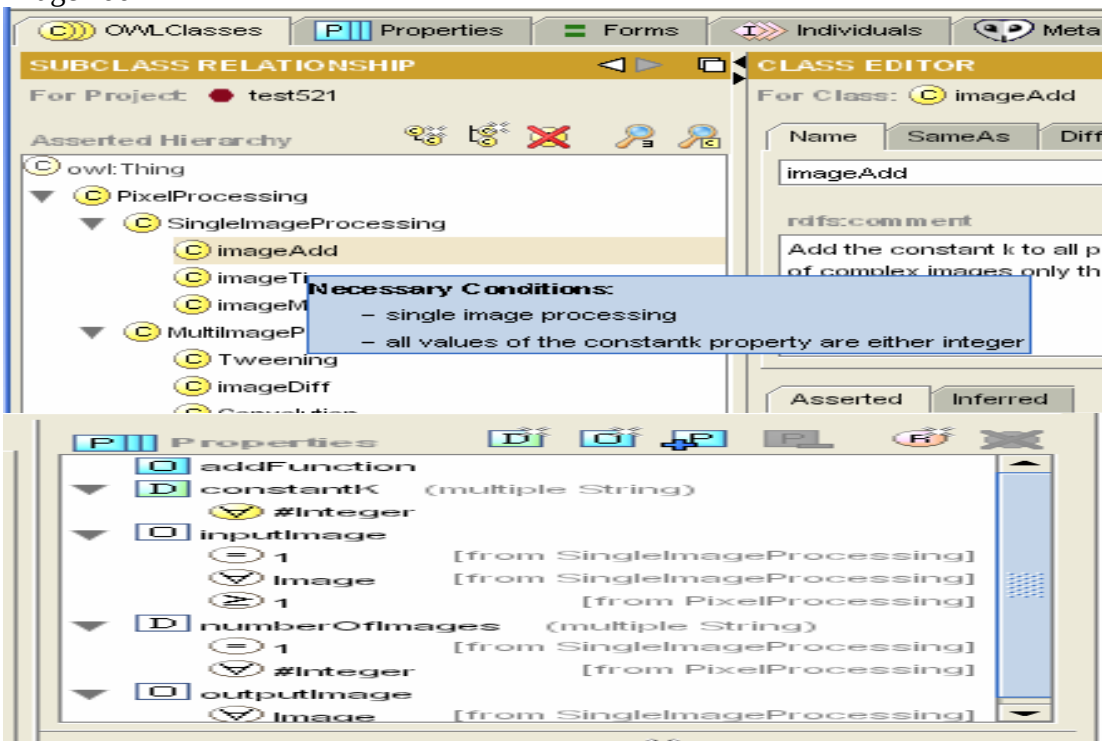




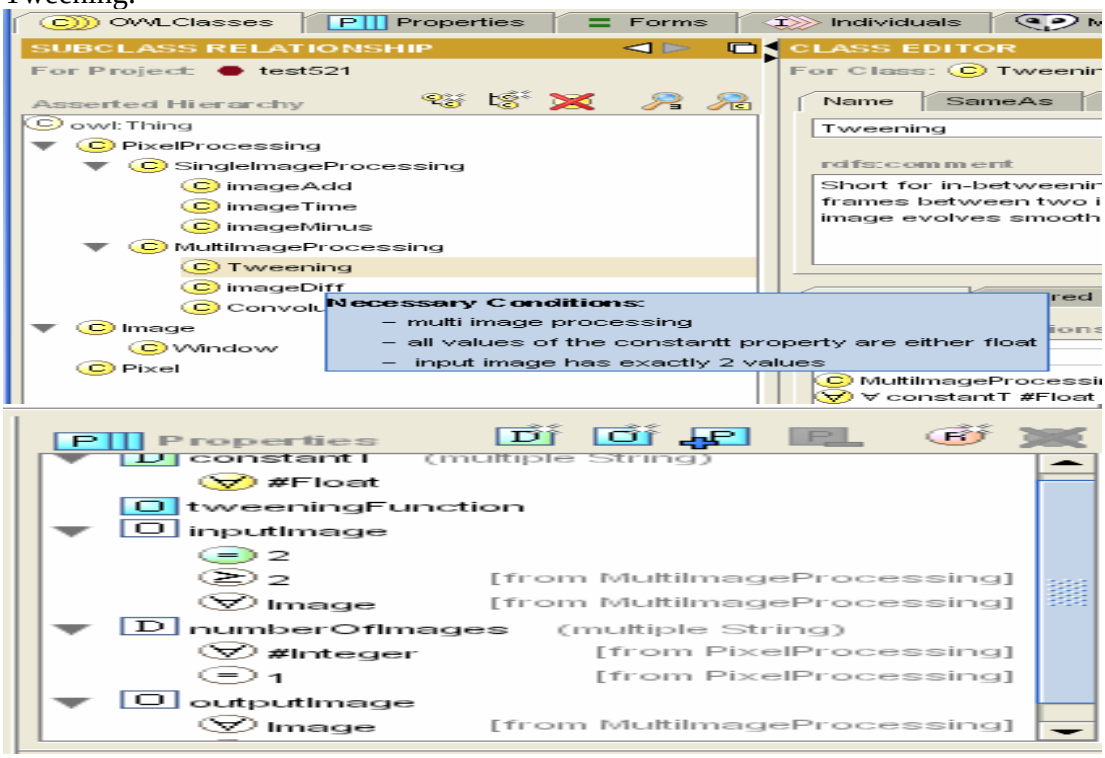
MultiImageProcessing


OWLClasses
Properties
Forms
Individuals
SUBCLASS RELATIONSHIP
For Project: test521
Asserted Hierarchy
owl:Thing
PixelProcessing
SingleImageProcessing
imageAdd
imageTime
imageMinus
MultiImageProcessing
Image
Pixel
Necessary Conditions:
– pixel processing
– input image has at least 2 values
– all values of the input image property are of type image
– all values of the output image property are of type image
– output image has exactly 1 value
CLASS EDITOR
For Class: MultiImageP
Name
SameAs
MultiImageProcessing
rdfs:comment
process involves 2 or mo
∀ inputImage Image
Properties
inputImage
Image
2
numberOfImages (multiple String)
#Integer [from PixelProcessing]
1 [from PixelProcessing]
outputImage
Image
1

imageAdd


OWLClasses
Properties
Forms
Individuals
SUBCLASS RELATIONSHIP
For Project: test521
Asserted Hierarchy
owl:Thing
PixelProcessing
SingleImageProcessing
imageAdd
Tweening
imageDiff
Necessary Conditions:
– single image processing
– all values of the constantk property are either integer
CLASS EDITOR
For Class: imageAdd
Name
SameAs
imageAdd
rdfs:comment
Add the constant k to all p
Asserted
Inferred
Properties
addFunction
constantK (multiple String)
#Integer
inputImage
1 [from SingleImageProcessing]
Image [from SingleImageProcessing]
1 [from PixelProcessing]
numberOfImages (multiple String)
1 [from SingleImageProcessing]
#Integer [from PixelProcessing]
outputImage
Image [from SingleImageProcessing]

Tweening:



4. Examples

Here are two examples of how is the instance of a class in OWL, and that instance is what may be used in some interface or application for intelligent search or reasoning, depending on different kind of implementation in fields like natural language understanding, program analysis, etc.

Tweening:
Short for in-betweening, the process of generating intermediate frames between two images to give the appearance that the first image evolves smoothly into the second image. Tweening is a key process in all types of animation, including computer animation. Sophisticated animation software enables you to identify specific objects in an image and define how they should move and change during the tweening process.
The algorithm for tweening is as follows [3]:

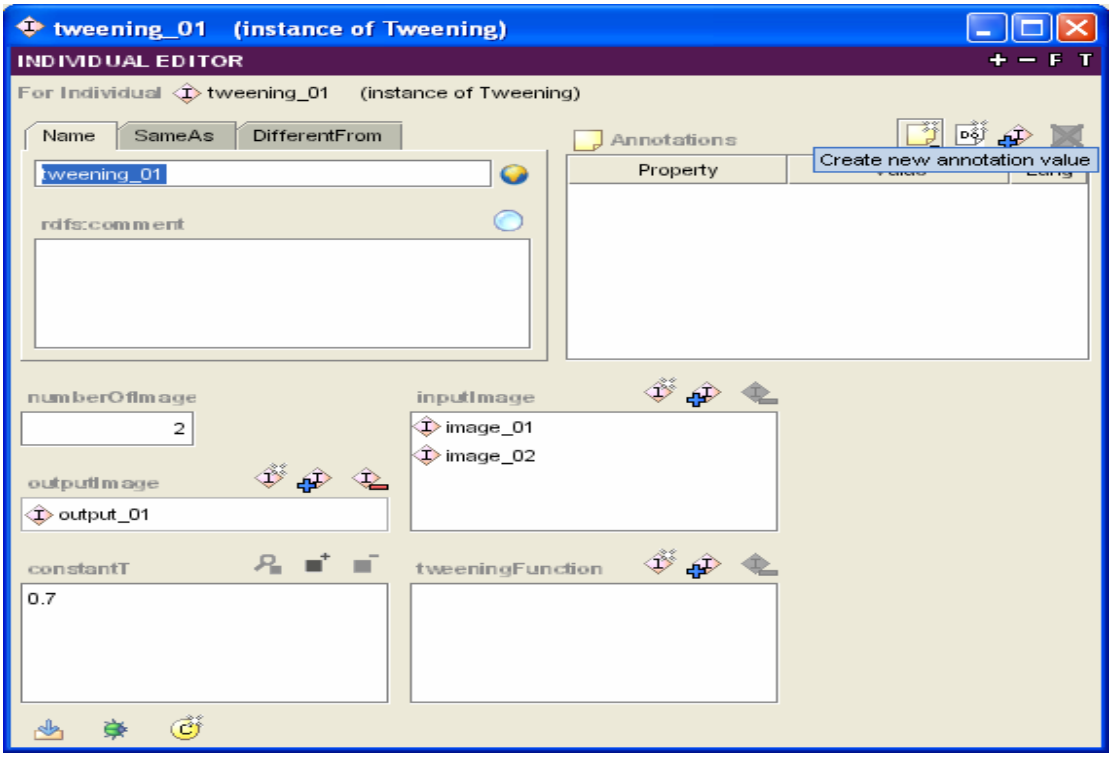
$$outputimage = (1-t) * inputimage1 + t * inputimage2$$

Images, whether input or output, are pixel matrices. However, input image 1 and 2 have to be of same dimension. That means, if image 1 is an 800*640 matrix, then image 2 has to be an 800*640 matrix also. "t" is a real number greater or equal to 0 and less or equal to 1.

The following section of code is OWL for the class of Tweening:

```
<owl:Class rdf:ID="Tweening">
    <rdfs:subClassOf rdf:resource="#MultiImageProcessing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#inputImage"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Short for in-betweening, the process of generating intermediate
frames between two images to give the appearance that the first image
evolves smoothly into the second image. </rdfs:comment>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="constantT"/>
        </owl:onProperty>
        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

There is an individual of Tweening class



Convolution:
Convolution, the mathematical, local operation is central to modern image processing. The basic idea is that a window of some finite size and shape--the support--is scanned across the image. The output pixel value is the weighted sum of the input pixels within the window where the weights are the values of the filter assigned to every pixel of the window itself. The window with its weights is called the convolution kernel. This leads directly to the following variation on eq. .

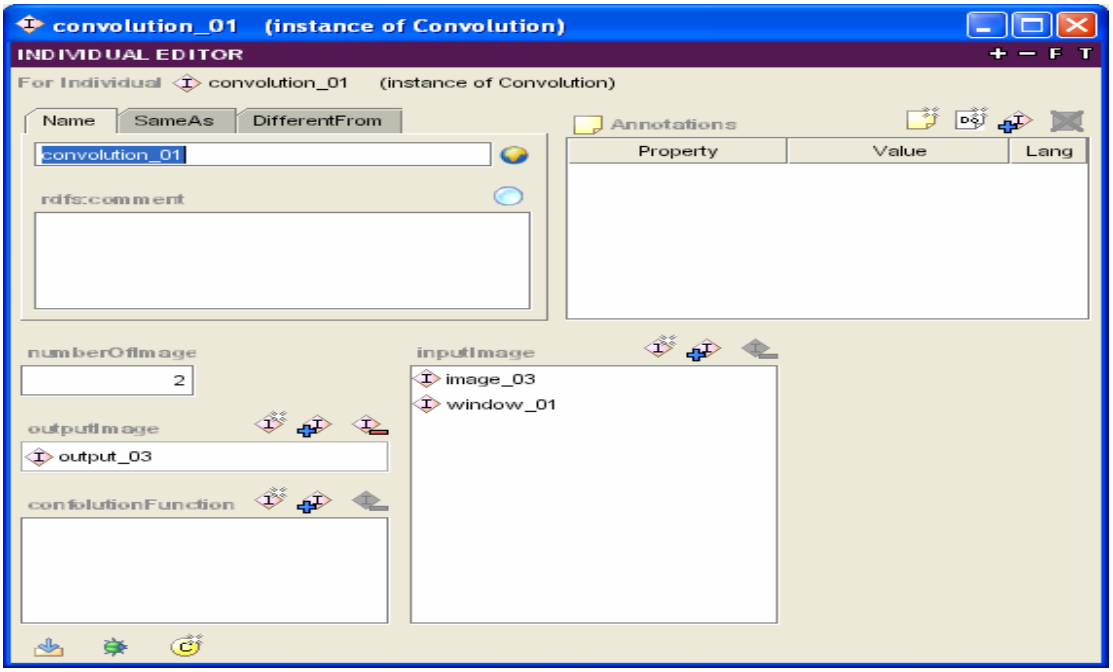
If the filter h[j,k] is zero outside the (rectangular) window {j=0,1,...,J-1; k=0,1,...,K-1}, then, using eq. , the convolution can be written as the following finite sum[3]:

$$c[m,n] = a[m,n] \otimes h[m,n] = \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} h[j,k] a[m-j, n-k]$$

Also, we have the code as follows:

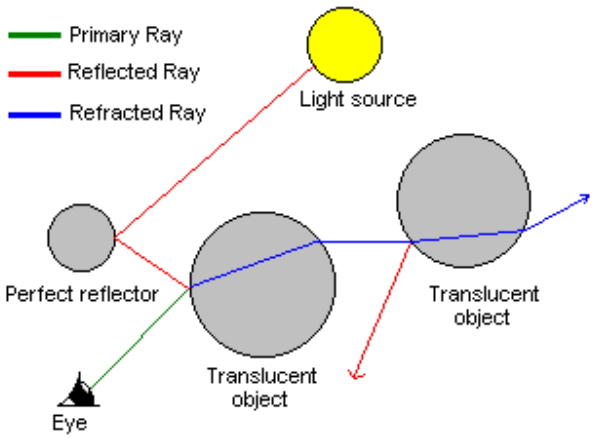
```
<owl:Class rdf:ID="Convolution">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >2</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="inputImage"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="MultiImageProcessing"/>
    </rdfs:subClassOf>
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >The basic idea is that a window of some finite size and shape--
the support--is scanned across the image. </rdfs:comment>
  </owl:Class>
```

And the instance of the class Convolution.



Raytracing:

After having explored 2 2-dimensional image-processing functions, let's have a look at a 3-dimensional computer graph technique—raytracing. In the very simplest terms, raytracing is a method for producing views of a virtual 3-dimensional scene on a computer [4]. A raytracing program calculates the illumination effects of a surface by tracking, or tracing, the path of a light ray as it bounces off or is refracted through the surface [4].



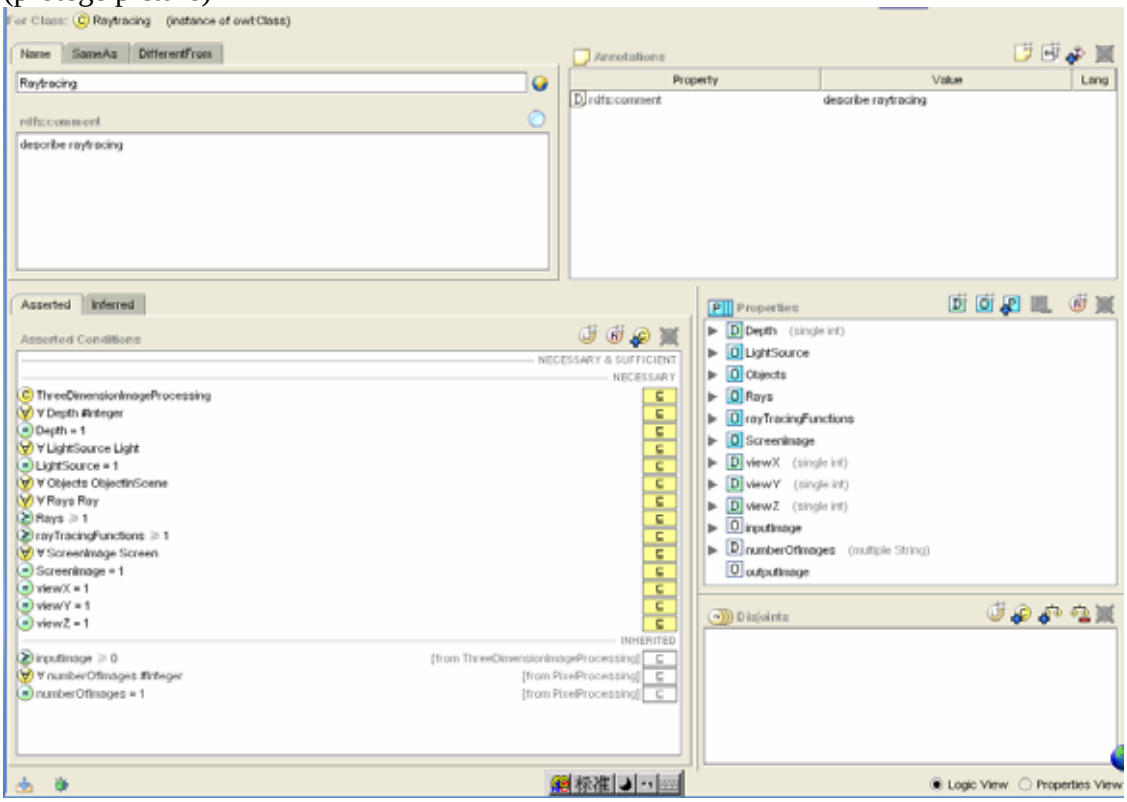
Let's describe the basic raytracing scenes. A scene in raytracing includes objects, light sources and viewpoint (also camera or eye), which is somehow similar with the picture above. In general, an object is any thing, either solid, liquid, or gas, that you will display in your scene [5]. Light sources, like objects, may be placed at arbitrary locations in the scene. However, unlike objects, light sources emit light [5]. In ray tracing, the viewpoint

or “camera” is much like this in that it determines where on the "film" (or, in the case of ray tracing, the computer screen) the light rays hit [4][5]. The basic process of generating the scene can be as follows. Taking out one ray emitting from the light source and follow that ray towards the objects, when the ray meets the proper object, there are two possibilities—one is reflecting, which means the surface of the object reflect the ray according to the mirror theory; the other is refracting, which means the object is transparent and the ray can be refracted through the object. Different objects will have different properties, so that the reflecting and refracting or both of them will happen according to the exact object the ray meets. Through out one or several such kind of actions among objects, the ray will finally reach the screen or the eye of a person taking the color or lightness or other appearance of each object and even the background. Those information will be calculated as pixel values for each pixel on the screen when display [5][6]. In realizing the raytracing scene into algorithm, people also reverse the process.

After the description of the scene and process of raytracing, we can have a look at the raytracing algorithm and the ray tracer. Actually, ray tracer is a raytracing program, which can perform the calculation of ray tracing. The algorithm begins, as in ray casting, by shooting a ray from the eye and through the screen, determining all the objects that intersect the ray, and finding the nearest of those intersections. It then *recurses* (or repeats itself) by shooting more rays from the point of intersection to see what objects are reflected at that point, what objects may be seen through the object at that point, which light sources are directly visible from that point, and so on [5]. Also, we could see that in such kind of pseudo-code as below [4]:

```
Procedure RenderPicture()
  For each pixel on the screen,
    Generate a ray R from the viewing position through the point on the view
      plane corresponding to this pixel.
    Call the procedure RayTrace() with the arguments R and 0
    Plot the pixel in the colour value returned by RayTrace()
  Next pixel
End Procedure
Procedure RayTrace(ray R, integer Depth) returns colour
  Set the numerical variable Dis to a maximum value
  Set the object pointer Obj to null
  For each object in the scene
    Calculate the distance (from the starting point of R) of the nearest
      intersection of R with the object in the forward direction
    If this distance is less than Dis
      Update Dis to this distance
      Set Obj to point to this object
    End if
  Next object
  If Obj is not null
    Set the position variable Pt to the nearest intersection point of R and Obj
    Set the total colour C to black
    For each light source in the scene
      For each object in the scene
        If this object blocks the light coming from the light source to Pt
          Attenuate the intensity of the received light by the transmittivity
            of the object
        End if
      Next object
      Calculate the perceived colour of Obj at Pt due to this light source
        using the value of the attenuated light intensity
      Add this colour value to C
    Next light source
    If Depth is less than a maximum value
      Generate two rays Refl and Refr in the reflected and refracted directions,
        starting from Pt
      Call RayTrace with arguments Refl and Depth + 1
      Add (the return value * reflectivity of Obj) to C
      Call RayTrace with arguments Refr and Depth + 1
      Add (the return value * transmittivity of Obj) to C
    End if
  Else
    Set the total colour C to the background colour
  End if
  Return C
End Procedure
```

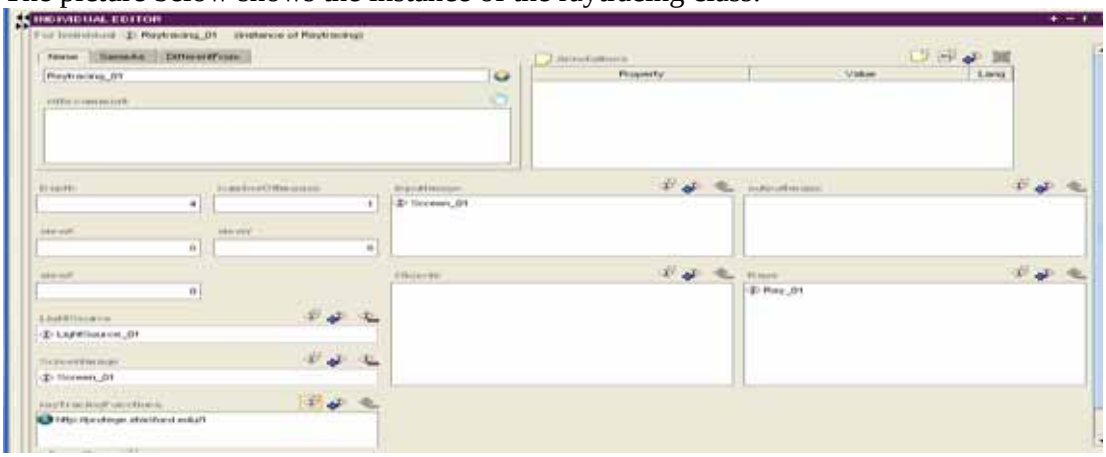
Based on the descriptions and the algorithms above, the simplified model of Raytracing class can be designed as below:
(protégé picture)



Here I made some assumptions to simplify the model:

1. Use point light source, so that don't need to consider the distribution of the light
2. Assume the whole scene in a 3-dimension coordinate system.
3. Use points on the boarder of the object and the center position of the object to locate the object and represent the shape of object.

The picture below shows the instance of the raytracing class.



5. Evaluations and Conclusion

According to the result from protégé, the ontology can satisfy the basic requirement of describing the pixel processing concepts. The most important thing is that, it can describe an action not just for concepts, in some logical way. However, the disadvantage is that it still needs some technique to enhance the ability to describe the restriction of processing (function) parameters. On the other hand, since ontology built by OWL can be read by computers, that makes the possibility that the ontology of pixel processing can be used in automatic programming or program analysis. Next step of this research is to perfect the whole ontology hierarchy and also develop an interface for other applications to get in text which related to pixel processing or even real pixel processing programs or algorithms. Natural language understanding or automatic program analysis and programming can be the ultimate goal.

6. Reference

1. Mostafa Aref, Zhengbo Zhou "The Ontology Web Language (OWL) for a Multi-Agent Understating System", IEEE 2005 International Conference on Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration, and Engineering, Waltham, MA, PP 586-591, 2005
2. http://www.tina-vision.net/manuals/prog_ref/node84.html
3. http://www.answers.com/topic/pixel
4. http://fuzzyphoton.tripod.com
5. http://www.geocities.com/jamisbuck/raytracing.html
6. Edward Angel "Interactive Computer Graphics: A Top-Down Approach Using OpenGL", Third Edition, Addison-Wesley Publishing Company, 2003
7. J.D Foley, A. Van Dam "Fundamentals of Interactive Computer Graphics" Addison-Wesley, 1982

The Ontology Web Language (OWL) for a Multi-Agent Understating System

Mostafa M. Aref Zhengbo Zhou
Department of Computer Science and Engineering
University of Bridgeport, Bridgeport, CT 06601
email: aref@bridgeport.edu

Abstract— *Computer understanding is a challenge problem in Artificial Intelligence. A multi-agent system has been developed to tackle this problem. Among its modules is its knowledge base (vocabulary agents). This paper discusses the use of the Ontology Web Language (OWL) to represent the knowledge base. An example of applying OWL in sentence understanding is given. Followed by an evaluation of OWL.*

1. INTRODUCTION

One of various definitions for Artificial Intelligence is "The study of how to make computers do things which, at the moment, people do better"[7]. From the definition of AI mentioned above, "Understanding" can be looked as the first step for a system to realize the ability of doing things as well as humans. Natural language processing needs an understanding system to make the machine understand human languages. Understanding is a transformation from one representation to another [1]. To achieve this transformation, the input will be processed through a series of agents. From morphological analysis to pragmatic analysis, the machine can "read" the input and has its own representation. Several applications may be developed based on the understanding system. Some examples of these applications are Machine learning, machine translating, and expert systems with better performance.

A multi-agents understanding system accepts a user input in a form of speech (typed or voice). Then, the user may enter several questions concerning the user input. The system should answer these questions that reflects the understanding of the input [1]. The multi-agents understanding system consists of the following agents: a morphological analyzer, a semantic analyzer, a discourse analyzer, a user interface, and a knowledge base. The knowledge base is the main module in the understanding system. It contains the English vocabulary agents and all the linguistic information about the vocabulary using object-oriented technology [1].

OWL is a Web Ontology Language. It is built on top of RDF – Resource Definition Framework and written in XML. It is a part of Semantic Web Vision, and is designed to be interpreted by computers, not for being read by people. OWL became a W3C (World Wide Web Consortium) Recommendation in February 2004 [2]. The OWL is a language for defining and instantiating *Web ontologies*. OWL ontology may include the descriptions of classes, properties, and their instances [3]. Given such ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but *entailed* by the semantics.

Section 2 describes a multi-agents understanding system. Section 3 gives a brief description of a newly standardized technique, Web Ontology Language—OWL. A working example of the OWL applied in knowledge representation is given in section 4. Section 5 evaluates the performance of OWL. Conclusion and directions of the current research are presented in section 6.

2. MULTI-AGENT UNDERSTANDING SYSTEM

To understand something is to transform it from input representation into internal representation has been chosen to correspond to a set of available actions that could be performed [1]. The process of natural language understanding is as follows [7], as shown in Figure 1.

Morphological Agent: given the input text, morphological analyzer converts the text into group of words in the basic form and their linguistic information. It also separates the affixes from the input tokens [1]. Semantic Agent: structures are created to represent meanings of a group of words (sentence). In other words, a mapping is made between the input sentence and objects in the task domain. Discourse Agent: Given the agent sub-societies of set of sentences, discourse analyzer agent resolves references between these sentences. The user interface is needed to facilitate the communication between the understanding system and the user [1]. For example, a web page containing several text input boxes can get input from a human and then gives another page or dialog box with the answer or some other actions.

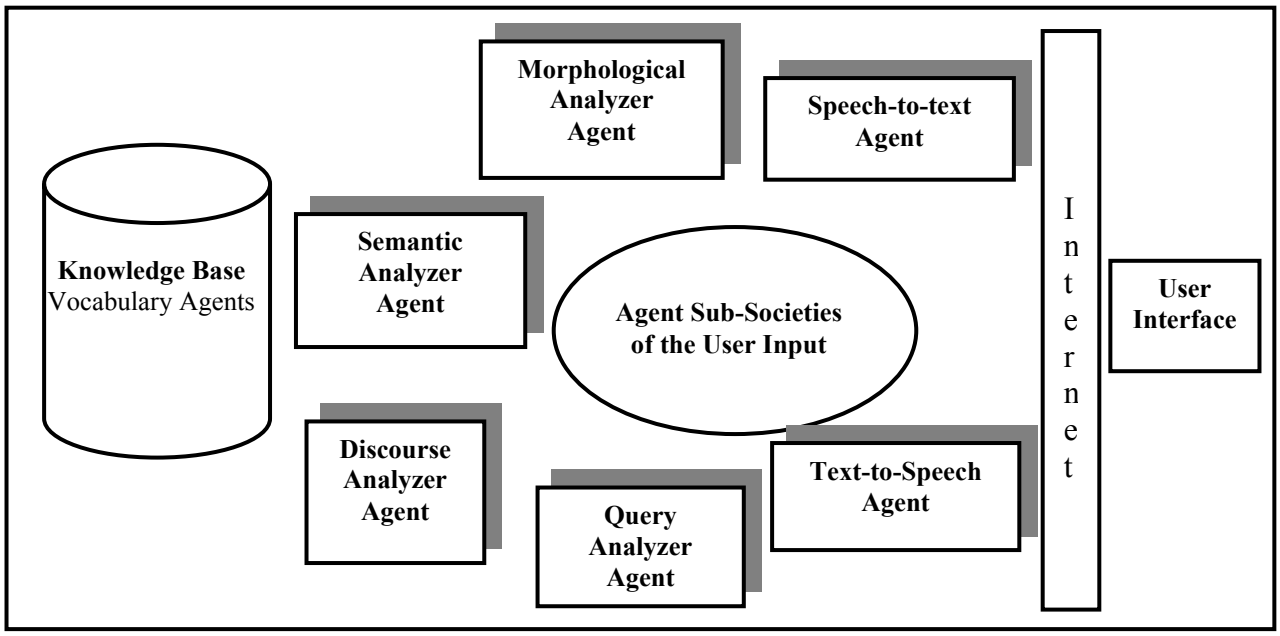


Figure 1 - Multi-agent understanding system

2.2 Knowledge Base

A knowledge base is a collection of knowledge expressed using some formal knowledge representation language [7]. In the understanding system, the knowledge base is the main module. It contains the English vocabulary agents and all linguistic information about this vocabulary. Good Knowledge representation is the basis of a good knowledge base. To evaluate one knowledge representation, there are the following four criteria [7]:

Representational Adequacy: the ability to represent all kinds of knowledge that are needed in that domain. Inferential Adequacy: the ability to manipulate the representational structures to derive new structures corresponding to new knowledge inferred from old. Inferential Efficiency: the ability to incorporate into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions. Acquisitional Efficiency: the ability to acquire new information easily. The simplest case involves direct insertion, by a person, of new knowledge into the database. Ideally, the program itself would be able to control knowledge acquisition.

The objective of knowledge representation is to organize the information necessary to the application such that it is easily accessed and manipulated. The knowledge content must be sufficient to solve problems in the domain and it must be efficient [1]. There are several knowledge representations such as: predicate logic, procedural, semantic nets, conceptual dependency and object-oriented representation [1] and [7].

Object-oriented knowledge representation organizes knowledge into classes of objects, subclasses and superclasses. That is an important issue in knowledge representation. By this organization, a class may inherit the properties of any of its superclasses and it may pass properties to any one of its subclasses [1]. However, only traditional object-oriented technique is not enough for a good knowledge representation or knowledge base.

3. OWL – ONTOLOGY WEB LANGUAGE

Ontology is about the exact description of things and their relationships. It is an old study of philosophy from ancient Greece. As the study of artificial intelligence growing, the concept of ontology have been using more and more in the formalization of knowledge in terms of classes, properties, instances and the relations. So, for knowledge, ontology is about the exact description of the representation of the knowledge itself, as well as the relationships among different categories of the knowledge. Moreover, for the web, ontology is about the exact description of web information and relationships between web information [2].

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. Because it provides additional vocabulary along with a formal semantics, OWL facilitates greater machine interpretability of Web content than XML, RDF, and RDF Schema (RDFS).

3.1 OWL Development

One of the effective approaches to solve the data exchange among different computers via the computer networks is using XML – eXtensible Markup Language. HTML is to solve how the data appears in front of people. The documents written in XML can describe what data is and be shared and exchanged among different systems [2]. XML provides a syntax for structured documents, however it imposes no semantic constraints on the meaning of the document. Following this, the W3 Consortium introduces the idea of ontology when creating RDF – the Resource Definition Framework. RDF is a data model for objects and relations between them, providing a simple semantic for this data model. RDF uses XML syntax to describe objects and relations in the data model [5].

After that, RDFS is developed to describe properties and classes of RDF resources, with a semantic for creating hierarchies of such objects and classes and thus providing the means for generalization. RDFS is considered to be an ontology language, containing classes and properties and being aware of concepts of range and domain, as well as having the ability describe subclasses and superclasses. However, for implementing the Semantic Web RDFS is not quite optimal as it lacks the features necessary to describe resources in sufficient detail. As Santtu Toivonen concludes in his research [9], RDFS is suitable for providing the means for an ontology that characterizes some environment, no matter how abstract [5]. RDFS alone, however, suffers from its dependence on domain-specific and case-specific details. RDFS suffers from an expressive inadequacy and it lacks a number of important relations between classes such as equivalence and disjointedness, as well as cardinality and characteristics of properties.

To solve the RDFS problems, two languages were developed almost concurrently. OIL (Ontology Inference Layer) in Europe and DAML (DARPA Agent Markup Language) in U.S.. Both of them are based on top of RDFS. After submitted the combination of the two—DAML+OIL to W3 Consortium for standardization, OWL –the Ontology Web Language is born as a new W3C standard language. OWL is layered on top of RDFS, using its syntax for expressing ontological primitives such as Class, Relation, Subclass etc. In addition OWL adds a much richer set of its own primitives, such as transitivity, cardinality, disjunction etc. Also, it adds characteristics of properties like symmetry, richer typing of properties (e.g. nonNegativeInteger), and enumerated classes [5]. As a result, OWL has more facilities for expressing meaning and semantics than XML and RDF (S). Thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. Figure 2 shows the development from XML to OWL.

The main purposes of OWL can be concluded as below:

1- Formalize a domain by defining classes and properties of those classes,
2- Define individuals and assert properties about them, and
3- Reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language.

3.2 The Species of OWL

All of these have led to a set of requirements that may seem incompatible: efficient reasoning support and convenience of expression for a language as powerful as a combination of RDF Schema with a full logic.

Indeed, these requirements have prompted W3C's Web Ontology Working Group to define OWL as three different sublanguages [4], each of which is geared towards fulfilling different aspects of this incompatible full set of requirements:

OWL Full—The entire language is called OWL Full, and uses all the OWL languages primitives. It also allows combining these primitives in arbitrary ways with RDF and RDF Schema. This includes the possibility (also present in RDF) to change the meaning of the pre-defined (RDF or OWL) primitives, by applying the language primitives to each other. For example, in OWL Full people could impose a cardinality constraint on the class of all classes, essentially limiting the number of classes that can be described in any ontology. The advantage of OWL Full is that it is fully upward compatible with RDF, both syntactically and semantically: any legal RDF document is also a legal OWL

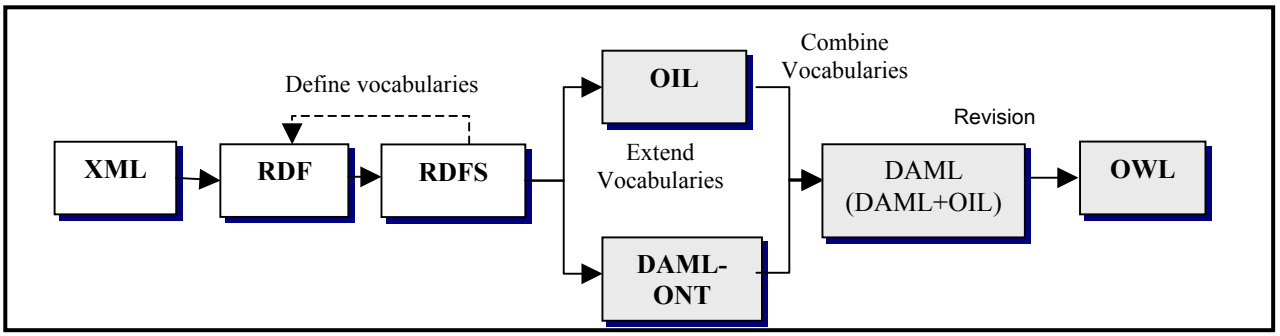


Figure 2 - the Development of OWL.

Full document, and any valid RDF/RDF Schema conclusion is also a valid OWL Full conclusion. The disadvantage of OWL Full is the language has become so powerful as to be undecidable, dashing any hope of complete (let alone efficient) reasoning support [3].

OWL DL (Description Logic)—In order to regain computational efficiency, OWL DL is a sublanguage of OWL Full which restricts the way in which the constructors from OWL and RDF can be used. Roughly this amounts to disallowing application of OWL's constructor's to each other, and thus ensuring that the language corresponds to well-studied description logic. The advantage of this is that it permits efficient reasoning support. The disadvantage is the lose of full compatibility with RDF. An RDF document will in general have to be extended in some ways and restricted in others before it is a legal OWL DL document. Conversely, every legal OWL DL document is still a legal RDF document [3].

OWL Lite—An ever further restriction limits OWL DL to a subset of the language constructors. For example, OWL Lite excludes enumerated classes, disjointness statements and arbitrary cardinality (among others). The advantage of this is a language that is both easier to grasp (for users) and easier to implement (for tool builders). The disadvantage is of course a restricted expressivity [3].

Ontology developers adopting OWL should consider which sublanguage best suits their needs. The choice between OWL Lite and OWL DL depends on the extent to which users require the more-expressive constructs provided by OWL DL and OWL Full. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modeling facilities of RDF Schema (e.g. defining classes of classes, or attaching properties to classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable since complete OWL Full implementations will be impossible.

There are strict notions of upward compatibility between these three sublanguages. Every legal OWL Lite ontology is a legal OWL DL ontology. Every legal OWL DL ontology is a legal OWL Full ontology. Every valid OWL Lite conclusion is a valid OWL DL conclusion. Every valid OWL DL conclusion is a valid OWL Full conclusion [3].

3.3 Structure and Basic Element of OWL

OWL as a Web Ontology Language can define classes using XML syntax. Figure 3 shows a simple name classes. Actually, people know almost nothing about these classes other than their existence, despite the use of familiar English terms as labels. Within this document, the Noun class can now be referred to using #Noun, e.g. rdf:resource="#Noun". Another form of reference uses the syntax rdf:about="#Noun" to extend the definition of a resource. It permits the extension of the imported definition from sources in other OWL construct without modifying the original document and supports the incremental construction of a larger ontology [3].

```
<owl:Class rdf:ID="Verb"/>
</owl:Class>
<owl:Class rdf:ID="Noun"/>
</owl:Class>
```

Figure 3 - Simple Named Classes

The fundamental taxonomic constructor for classes is rdfs:subClassOf. It relates a more specific class to a more general class. If X is a subclass of Y, then every instance of X is also an instance of Y. The rdfs:subClassOf relation is transitive. If X is a subclass of Y and Y a subclass of Z then X is a subclass of Z. Apparently, with the definition of a class and a subclass, we can apply "is a" relationship very easily by OWL.

Individuals— In addition to classes, it is possible to be able to describe their members. We normally think of these as individuals in our universe of things. An individual is minimally introduced by declaring it to be a member of a class e.g. <Verb rdf:ID="buy1" />.

"rdf:type" is an RDF property that ties an individual to a class of which it is a member. There are a couple of points to be made here. First, it has already decided that "buy" (a specific verb) is member of Verb, the class containing all kinds of verbs. Second, there is no requirement in the two-part example that the two elements need to be adjacent to one another, or even in the same file (though the names would need to be extended with a URI in such a case). People design Web ontologies to be distributed. They can be imported and augmented, creating derived ontologies. Figure 4 shows an example of a subclass and an individual.

```
<owl:Class rdf:ID="CountableNoun">
<rdfs:subClassOf rdf:resource="#Noun" />
</owl:Class>
<owl:Verb rdf:ID="buy1" />
<rdf:type rdf:resource="#Verb"/>
</owl:Thing>
```

Figure 4 - A Subclass and An Individual

Simple Properties—This world of classes and individuals would be pretty uninteresting if there is the only definition of taxonomies. Properties let us assert general facts about the members of classes and specific facts about individuals [3]. A property is a binary relation. Two types of properties are distinguished [3]. Datatype properties: relations between instances of classes and RDF literals and XML Schema datatypes. Object properties: relations between instances of two classes. Figure 5 shows an object property example. From the given above, it is not only to know that "raise" is transitive, but also it is able to infer from the domain that "raise" is a verb.

Data types [3]— OWL uses most of the built-in XML Schema datatypes. References to these datatypes are by means of the URI reference for the datatypes [2]. Now OWL is still under testing and study, the effectiveness of its expression and communication among machines attracts the experts in related area to use this language and do research with it. Next section is going to present current implementations.

```
<owl:ObjectProperty rdf:ID="isTransitive">
<rdfs:domain rdf:resource="#Verb"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Word"/>
</owl:Class>
<owl:Class rdf:ID="Verb">
<rdfs:subClassOf rdf:resource="#Word" />
</owl:Class>
<owl:Word rdf:ID="Raise">
<isTransitive/>
</owl:Word>
```

Figure 5 - Object Property Example

3.4 Linguistic OWL

To use OWL in the understanding system knowledge base, it is by first to gather linguistic information. OWL version of Wordnet is a project that translate the Wordnet database into OWL. There is an ongoing project called "Wordnet in RDFS and OWL" [10]. It is one of the Semantic Web Best Practices, and developed by Wordnet Task Force.

One of its approaches is integrating existing datamodels in order to provide a unified OWL vocabulary for RDF versions of Wordnet [11]. There is a WordNet.OWL which is an OWL-ontology based on WordNet 1.7.1 [12]. Also, a little bit earlier, there is another project to develop ontologies known as SUMO—Suggested Upper Merged Ontology, and now there is also this version (partial) of Wordnet database [13]. This ontology is being created as part of the IEEE Standard Upper Ontology Working Group. The goal of this Working Group is to develop a standard upper ontology that will promote data interoperability, information search and retrieval, automated inferring, and natural language processing [13].

4. OWL IN KNOWLEDGE BASE

To illustrate the working of OWL in knowledge representation, an example of a text input is given.

Chris bought an old car with $5000 last Friday.

- The verb: "buy"
- Verb's agent: "Chris"
- Result of the action: "car"
- Time of the action: "last Friday"
- Money in the transaction: "$5000"

According to the requirement of knowledge representation of natural language, there is the corresponding OWL code for the content in knowledge base: In the figure 6, "agent", "transactionAmount", "time" and "result" are attributes of object "buy1".

```
<Buying rdf:ID="buy1">
<agent rdf:resource="#Agent" />
<agent rdf:ID="Chris" />
<transactionAmount rdf:resource="#CurrencyMeasure" />
<transactionAmount rdf:ID="$5000" />
<time rdf:resource="#TimePosition"/>
<time rdf:ID="Friday"/>
<result rdf:resource="#Entity"/>
<result rdf:ID="car1"/>
</Buying>
<Human rdf:ID="Chris">
<immediateInstance rdf:resource="#Human"/>
</Human>
<Vehicle rdf:ID="car1">
<monetaryValue rdf:resource="#CurrencyMeasure"/>
<monetaryValue rdf:ID="$5000"/>
<property rdf:resource="#Attributes"/>
</Vehicle>
<UnitedStatesDollar rdf:ID="$5000">
<lessThanOrEqualTo rdf:datatype="&xsd;string">5000
< /lessThanOrEqualTo>
<greaterThanOrEqualTo rdf:datatype="&xsd;string">5000
</greaterThanOrEqualTo>
</UnitedStatesDollar>
<Friday rdf:ID="Friday1">
<PastFn rdf:resource="#TimePosition" />
</Friday>
```

Figure 6 - An Example in OWL

5. EVALUATION OF OWL

The four criteria to evaluate a proper knowledge representation are representational adequacy, inferential adequacy, inferential efficiency and acquisitional efficiency. However, unfortunately, up to now no single system that optimizes all of the capabilities for all kinds of knowledge has yet been found. Following these four principles, we will see how OWL can work for the knowledge representation in a natural language processing system.

Since OWL is using XML, it has a strong ability that can be shared and exchanged between different types of computers using different types of operating system and application languages.

Representational adequacy—By building up from XML, OWL inherits its main function of describing data. By using ontology in XML syntax and necessary RDF Schema, OWL describes the domain knowledge in ontology primitives (objects, classes, properties etc). Basically, by using this kind of representation, inheritance can be

performed efficiently using ontologies. As from examples in section 3.3, the property "subclassof" can explicitly describe the relation. Furthermore, the whole fact set of OWL for classes and objects include the description of an ontology, and the axioms describes the manipulation on each ontology. As a result, it is not only able to describe things in detail, but also able to provide a large base of relationships and the probability of doing reasoning among different ontologies.

Inferential adequacy—As mentioned above, axiom can play a role of knowledge operator. Axioms are used to associate class and property identifiers with either partial or complete specifications of their characteristics, and to give other information about classes and properties. Axioms used to be called definitions, but they are not all definitions in the common sense of the term and thus a more neutral name has been chosen [5]. Besides axioms, OWL also has other kind of manipulation on ontologies.

Acquisitional efficiency—Compared to some of the knowledge representation method mentioned in [7], ontologies developed by OWL have their own advantages in this criterion. But, as deputed by a lot of researchers, this kind of knowledge representation still lacks of acquisitional efficiency. Most of the ontologies are created by people, not by machine.

Disadvantages—So far, it may not be possible to describe OWL semantics with logic programs or rule base directly. There should be some tools, including reasoners, validators or so to do some ancillary work during using OWL. Another inconvenience is, as a new w3c standard, it is not prevailed in a very wide range, and a lot of research work and testing work are undergoing to exploit its usage. Only few organizations, including university laboratories and academic organizations, are studying or implementing OWL. Similar to other knowledge representation technique, it is not easy for OWL to have good performance on knowledge acquisition. However, OWL is still being studied and developed, so, probably it is possible to find out the breakthrough of this principle in OWL in the future.

6. CONCLUSION AND FUTURE TASK

Natural language understanding may be considered as being a mapping of a natural language text to an internal representation that capture the meaning of that text. This paper presents an ongoing research project about a multi-agents understanding system and the construction of its knowledge base. The knowledge base is the foundation of the work and communication between each of the modules in the system. The new technique of OWL—Web Ontology Language, provides a new tool to implement knowledge representation. Some of its feature fits the requirement of knowledge representation and are proved to be efficient by some tests and implementations. However, as a new standard and language, there is the opportunity to develop its potential ability in knowledge representation and natural language processing.

After explored the OWL and evaluated its ability in knowledge representation, the current research focuses on its implementation in this area and discuss its ability in machine learning based on the NLP with OWL. To do this research, WordNet provides a certain kind of platform and foundation. A task force of semantic web is doing the work to convert the WordNet database into OWL.

REFERENCE

[1] Mostafa Aref, "A Multi-Agent System for Natural Language Understanding", Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 1-3, Cambridge MA, PP 36-40, 2003.

[2] W3C School, "Tutorial of RDF OWL", www.w3schools.com, 2004.

[3] Michael K. Smith, Chris Welty, Deborah L. McGuinness, "OWL Web Ontology Language Guide", W3C Recommendation, www.w3.org/TR/owl-guide, 10 February 2004.

[4] Deborah L. McGuinness, Frank van Harmelen, "OWL Web Ontology Language Overview", W3C Recommendation, www.w3.org/TR/2003/PR-owl-features-20031215/#ref-rdf-schema, 10 February 2004.

[5] Kostyantyn Vovk, Ontology and Object-Oriented Representation, Spring 2004.

[6] Sandro Hawke, "OWL Implementation", http://www.w3.org/2001/sw/WebOnt/impls, 2003.

[7] Elaine Rich, Kevin Knight, Artificial Intelligence, Second Edition 1991 McGraw-Hill Inc.

[8] Stuart Russell, Peter Norvig Artificial Intelligence: A Modern Approach, 1995 Prentice Hall ; Pearson Education.

[9] S. Toivonen. "Using RDF(S) to Provide Multiple Views into a Single Ontology". SemWeb 2001 Workshop, Hong Kong, 2001.

[10] "Wordnet in RDFS and OWL", http://www.w3.org/2001/sw/BestPractices/WNET/wordnet-sw-20040713.html W3C 2004.

[11] Coordinator Aldo Gangemi, "Semantic Web Best Practices", WordNet Task Force http://www.w3.org/2001/sw/BestPractices/WNET/tf, Wordnet Task Force, 2004.

[12] KID Processing Group, "knOWLer: Ontology-based Information Management System", http://taurus.unine.ch/knowler/ 2004.

[13] Deborah Nichols and Allan Terry, "SUMO Ontology", http://ontology.teknowledge.com/ Teknowledge Corp.2004.